

A Framework for Automated Software Testing and Defect Management: Integrating Machine Learning for Intelligent Bug Triaging and Root Cause Identification

Urvish Gajjar¹

¹ sr test manager, health care service corporation.

Article Info

Article History:

Published: 19 June 2026

Publication Issue:

Volume 3, Issue 6
June-2026

Page Number:

1056-1059

Corresponding Author:

Urvish Gajjar

Abstract:

The increasing complexity and rapid release cycles in modern software development demand sophisticated approaches to software testing and defect management. Traditional methods, while foundational, often struggle to keep pace with the volume and intricacies of bugs encountered. This article proposes a novel framework that integrates machine learning (ML) techniques to enhance automated software testing and defect management. The framework focuses on intelligent bug triaging and automated root cause identification, aiming to improve efficiency, reduce manual effort, and accelerate the software development lifecycle.

Keywords: Software Testing, Defect Management

1. INTRODUCTION

Software development has evolved dramatically, with agile methodologies and continuous integration/continuous deployment (CI/CD) pipelines becoming standard practices. This acceleration, while beneficial for rapid delivery, has amplified the challenges in ensuring software quality. The sheer volume of test cases, execution logs, and bug reports generated can overwhelm manual processes [1]. Defect management, in particular, requires significant human effort for bug triaging (assigning bugs to the right teams or individuals) and root cause analysis (identifying the underlying reasons for defects), which are often time-consuming and prone to subjective biases [2].

To address these challenges, this paper presents a comprehensive framework that leverages machine learning (ML) to automate and enhance key aspects of software testing and defect management. The proposed framework integrates ML models for intelligent bug triaging and automated root cause identification, aiming to streamline the defect resolution process, improve accuracy, and ultimately deliver higher-quality software more efficiently. The framework is designed to be adaptable and can be integrated into existing development workflows.

2. RELATED WORK

Automated software testing has seen significant advancements, with tools and techniques evolving to cover various testing levels, from unit to system testing. Machine learning has also begun to permeate software engineering practices. Early applications of ML in testing include test case prioritization [3], test selection [4], and defect prediction [5]. In defect management, ML has been explored for predicting defect severity and for identifying duplicate bug reports [6]. However, a unified framework that synergistically combines intelligent bug triaging and automated root cause analysis using ML within an integrated testing and defect management system remains an area ripe for development.

3. THE PROPOSED FRAMEWORK

The proposed framework is designed to augment existing software testing and defect management processes by introducing intelligent automation powered by machine learning. It consists of several interconnected modules:

A. Automated Test Execution and Data Collection

This module encompasses the execution of automated tests (unit, integration, system, etc.) and the systematic collection of relevant data. This data includes:

- Test case execution results (pass/fail, execution time).
- System logs and error messages generated during test runs.
- Code commit history and associated metadata (author, timestamp, files changed).
- Existing bug reports and their resolution details (assigned team, root cause, fix).

This data serves as the foundation for training and operating the ML models within the framework.

B. Intelligent Bug Triaging Module

This module utilizes ML models to automatically assign incoming bug reports to the most appropriate development team or individual. The process involves:

- **Feature Engineering:** Extracting relevant features from bug reports, such as keywords in the title and description, error message patterns, affected components (if identifiable), and similarity to previously triaged bugs.
- **Model Training:** Training classification models (e.g., Support Vector Machines, Naive Bayes, deep learning models like LSTMs for text analysis) on historical bug report data, where each bug is labeled with its assigned team.
- **Automated Triaging:** Once trained, the model predicts the likely assignee for new bug reports based on the extracted features.

This significantly reduces the manual effort and time involved in the initial bug assignment process.

C. Root Cause Identification Module

This module aims to automate the identification of the likely root cause of a defect. It leverages a combination of techniques:

- **Log Analysis:** ML algorithms (e.g., anomaly detection, sequence analysis) are used to parse and analyze system logs associated with a failing test case or a reported bug to pinpoint suspicious error patterns or sequences of events.
- **Code Change Analysis:** By correlating bug occurrences with recent code commits in the affected modules, ML models can identify potentially problematic code changes. Techniques like feature importance from tree-based models can highlight the most likely contributing code modifications.
- **Historical Defect Correlation:** The system can identify similarities between the current defect and previously resolved defects, suggesting similar root causes or code areas.

The output of this module provides developers with actionable insights, narrowing down the search space for the actual bug fix.

D. Feedback Loop and Model Retraining

Continuous improvement is vital. This module ensures that the system learns from new data and human feedback:

- **Human Feedback Integration:** When developers manually correct a bug's triage assignment or identify a different root cause than predicted, this feedback is captured.

- **Model Retraining:** The ML models are periodically retrained with updated datasets, incorporating new bug reports, code changes, and human feedback to improve their accuracy and adapt to evolving codebases and defect patterns.

This closed-loop system ensures the framework remains effective over time.

4. ARCHITECTURE AND WORKFLOW

The framework integrates seamlessly into a typical CI/CD pipeline. The workflow is as follows:

1. Automated tests are executed.
2. Upon test failure or a new bug report, relevant data (logs, stack traces, commit history) is collected.
3. The Intelligent Bug Triaging module analyzes the report and assigns it to a team.
4. The Root Cause Identification module analyzes logs and code changes to suggest potential causes.
5. Developers receive the bug report with suggested triage and potential root causes.
6. Developers investigate, fix the bug, and provide feedback on the triage and root cause analysis.
7. This feedback is used to retrain the ML models, closing the loop.

The overall architecture is depicted in the following diagram:

Component	Description	ML Techniques Used
Automated Execution	Test Runs tests, collects results, logs, and metadata.	N/A
Data Collector	Gathers test results, logs, commit history, bug reports.	N/A
Intelligent Triaging	Bug Assigns bugs to appropriate teams.	Text Classification (SVM, Naive Bayes), Sequence Models (LSTM)
Root Identification	Cause Suggests potential causes for defects.	Anomaly Detection, Sequence Analysis, Feature Importance (Tree Models)
Feedback Loop	Captures human corrections and insights.	N/A
Model Retraining	Periodically updates ML models with new data.	N/A
Developer Interface	Presents bug reports with ML-driven suggestions.	N/A

Fig. 1. High-level architecture of the ML-integrated Software Testing and Defect Management Framework.

5. BENEFITS AND ADVANTAGES

Implementing this framework offers several advantages:

- **Increased Efficiency:** Automates time-consuming tasks of bug triaging and initial root cause analysis.
- **Reduced Manual Effort:** Frees up developers and QA engineers to focus on more complex problem-solving and strategic testing.
- **Faster Defect Resolution:** Quicker triaging and identification of potential causes lead to faster bug fixes.
- **Improved Accuracy:** ML models can identify patterns and correlations that might be missed by human analysis, leading to more accurate assignments and suggestions.
- **Enhanced Learning and Adaptability:** The feedback loop ensures the system continuously improves and adapts to the project's evolving nature.
- **Data-Driven Insights:** Provides valuable insights into defect patterns, common causes, and team performance.

6. CHALLENGES AND FUTURE WORK

Despite its potential, challenges exist. The effectiveness of ML models heavily relies on the quality and quantity of historical data. Cold-start problems (new projects with limited data) need to be addressed. Model interpretability can also be a concern, especially for deep learning models. Future work includes exploring advanced ML techniques for more nuanced defect prediction, integrating performance-related defects, and developing more sophisticated explainability features for the root cause identification module.

7. CONCLUSION

This paper proposed a framework for automated software testing and defect management that integrates machine learning for intelligent bug triaging and root cause identification. By leveraging ML, the framework aims to significantly enhance the efficiency and effectiveness of the defect resolution process. The automated analysis of bug reports, logs, and code changes provides developers with actionable insights, accelerating the delivery of higher-quality software. As ML technologies continue to mature, their role in transforming traditional software engineering practices will undoubtedly grow, making frameworks like this essential for modern development teams.

References

1. [1] A. S. Tan, "The Impact of Agile Development on Software Testing," *IEEE Software*, vol. 30, no. 4, pp. 78-81, Jul./Aug. 2013.
2. [2] S. R. K. Reddy and P. V. S. Kumar, "Challenges in Bug Triage: A Survey," *Proc. Int. Conf. on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1575-1580, 2014.
3. [3] T. Menzies, J. Orgun, and H. Yang, "Prioritizing Software Tests Using Machine Learning," *Proc. ACM SIGSOFT Symp. on Software Testing, Analysis, and Verification (ISSTA)*, pp. 139-149, 2007.
4. [4] B. S. Kalgotra, A. Kumar, and S. Singh, "Machine Learning Approaches for Test Case Selection: A Survey," *ACM Comput. Surv.*, vol. 53, no. 3, Article 54, pp. 1-36, Jun. 2020